HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Automation and Systems Technology
Degree Programme of Computer Science and Engineering

An e-business solution for the stock market insider registry information

Master's Thesis

Tero Hilska

Information and Computer Systems in Automation
Espoo 2006

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Automation and Systems Technology
Degree Programme of Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author: Tero Hilska

Title of thesis:
An e-business solution for the stock market insider registry information

Date: May 22nd 2006 **Pages:** 11 + 62
Professorship: Information Technology in Automation **Code:** AS-116
Supervisor: Professor Kari Koskinen
Instructor: Petri Leinonen, MSc (Econ.)

Modern e-Business is about integrating information systems both within and between organizations. Since there are many different kinds of systems, all accepting different kinds of messages as inputs and outputs, dedicated integration platforms are needed. Most integration platforms take advantage of Extensible Markup Language (XML) family, which is becoming a standard tool used for exchanging documents and messages.

In this thesis one of such platforms, Microsoft Biztalk Server 2004, is presented. The XML language family and some of its members are also described as needed for the practical part of this thesis.

The BizTalk Server is evaluated by using it to implement an Investor Relations (IR) service application. The application is intended to be used to meet the new requirements of publishing the insider information in the Internet. To fully understand the business field operated in, the key concepts of investor relations and insider information are defined.

Keywords: stock market, insider register, investor relations e-business, XML, Microsoft BizTalk, .NET
Language: English

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ
Automaatio ja Systeemitekniikan Osasto
Tietotekniikan Koulutusohjelma

Tekijä:	Tero Hilska	
Työn nimi:	An e-business solution for the stock market insider registry information	
Päiväys:	22. toukokuuta 2006	**Sivumäärä:** 11 + 62
Professuuri:	Automaation Tietotekniikka	**Koodi:** AS-116
Työn valvoja:	prof. Kari Koskinen	
Työn ohjaaja:	KTM Petri Leinonen	

Moderni sähköinen liiketoiminta on paljolti tietojärjestelmien integrointia sekä organisaatioiden sisällä että niiden välillä. Koska erilaisia tietojärjestelmiä on runsaasti ja niiden ymmärtämät syötteet ja tulosteet eroavat toisistaan, on kehitetty erityisiä integraatioalustoja. Monet integraatioalustat hyödyntävät Extensible Markup Language (XML) kieliperhettä, josta on tulossa vakiotyökalu dokumenttien ja viestien vaihdossa.

Tässä diplomityössä on esitelty yksi tälläinen integraatioalusta, Microsoft BizTalk Server 2004. XML kieliperhettä ja sen joitakin jäseniä kuvataan siinä laajudessa kuin on tarpeellista tämän diplomityön käytännön osuuden kannalta.

BizTalk Server integraatioalustaa arvioidaan käyttämällä sitä sijoittajaviestintä (IR) sovelluksen kehityksessä. Sovelluksen on tarkoitus täyttää uudet vaatimukset sisäpiirirekisteritietojen julkaisemisesta Internetissä. Liiketoimintaympäristön ymmärtämiseksi sijoittajaviestinnän ja sisäpiiritiedon keskeiset käsitteet on määritelty.

Avainsanat:	arvopaperipörssi, sisäpiirirekisteri, sijoittajaviestintä sähköinen liiketoiminta, XML, Microsoft BizTalk, .NET
Kieli:	Englanti

Acknowledgements

I would like to thank my former employer, who wants to remain anonymous, for the possibility to write this thesis in connection with a real software development project and understanding my urge to graduate.

Thanks to both Petri Leinonen, my instructor and Mika Ahorinta, who was not an instructor in paper, but still gave many valuable pieces of advice.

I would also like to thank my professor, Kari Koskinen, for understanding, when the thesis was not completed in the scheduled timeframe and encouraging me to finnish it.

Special thanks go to researcher Satu Elisa Schaeffer for reading my thesis through during a flight from Frankfurt to Helsinki and giving most valuable comments. This was perhaps the best guidance I received for this thesis.

Thanks to my family and friends for support and encouragement, especially for my little brother for acting as a proofreader. Reading technical text must have been frustrating for a student of marketing.

Finally, thanks to everlasting God, the Father, the Son and the Holy Ghost.

Espoo May 22nd 2006

Tero Hilska

Abbreviations and Acronyms

APK	Finnish Central Securities Depository
B2B	Business To Business
CLR	Common Language Runtime
CSV	Comma Separated Values
DSSSL	Document Style Semantics and Specification Language
DTD	Document Type Definition
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
ERP	Enterprise Resource Planning
GAC	Global Assembly Cache
HAT	Health and Activity Tracking
HWS	Human Workflow Services
IR	Investor Relations
IRI	Internationalized Resource Identifier
MSIL	Microsoft Intermediate Language
SGML	Standard Generalized Markup Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XDR	XML Data Reduced
XML	Extensible Mark-Up Language
XSL	Extensible Stylesheet Language
XSL-FO	Extensible Stylesheet Language - Formatting Objects
XSLT	Extensible Stylesheet Language: Transformations

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Trading stocks and other equities is basically trading future expectations. The price of an equity is affected by numerous things: the overall development of the market in general and the industry in particular, the sales, the profits, the expected dividends, a crisis in the Persian Gulf and so on. Companies usually want, and are even required, to report this information to their owners and creditors by best means possible.

Among those listed above, an investor nowadays usually considers the ownership structure of a company when making an investment decision: who owns the company, who holds the voting power, what portion of the company is the key management and how are those ownership structures changing. For example, if majority shareholders are bailing out, the company may not be the best investment to consider. On the other hand the CEO of the company holding on to his/her shares in spite of bad quarterly figures may give indication about his/her trust in the future.

Things are, of course, not so straightforward. Majority holders cashing in their shares may simply be in the need of some extra cash. The CEO may simply be following company's policy in order to reassure investors. But anyway, information about ownership structure and changes in it is valuable to the potential investor.

When shares and other securities were in the paper form and were exchanged from hand to hand, there was actually no way of knowing who held them at a certain moment. Nowadays, after the advent of book-entry securities accounts and certificate-less system, this information does finally exist. Individual account keepers (banks and securities associations) have information about holdings of their clients and there is usually also a central registry holder, in Finland the Finnish Central Securities Depository (APK).

But the existence of the information does not yet guarantee its availability and usefulness to the investors. Companies would like to publish the lists of their largest owner on their web site, not on some terminal in a bank. The information is there, but the challenge is to publish it. Nowadays, when trading of equities is sometimes quite hectic, nobody has the time to manually update an owner list on a web site, there must be a way to do this automatically. Thus, there is a need for an application that handles and publishes shareholder information.

The goal of this thesis is to develop such an application. The task is approached by building an e-Business solution. In the process we discuss about e-Business in general and the suitability of the Microsoft BizTalk Server in developing such applications.

In chapter 2 we take a look into some key concepts of investing and investor relations services. Also the recent modifications of the Securities Market Act are discussed. In the end, the objectives of this thesis are defined.

Chapter 3 is about the e-Business, the history and definition of the concept. Special consideration is given to the Electronic Data Interchange (EDI). Also the objectives are evaluated with respect to the concept of e-Business and EDI.

Extensible Markup Language (XML) family is discussed in chapter 4. The two members of that family, XSD and XLST, that are important for this thesis, are examined in detail.

Next chapter, number 5, presents the Microsoft BizTalk Server 2004, the integration platform that is used for implementing the application. The key elements of the BizTalk Server are examined.

Chapter 6 describes the implementation process, from the first plans to the actual implementation as well as the drawbacks encountered during the process.

After that, in chapter 7, the outcome of the implementation process is evaluated against the objectives defined in chapter 2.

The final chapter summarizes the contents and achievements of this thesis.

Chapter 2

Background

This thesis aims in designing and developing a new application for automated processing and publishing insider and shareholder registry information. The need for this work rises from both the new content requirements that the old application fails to full-fill and the efficiency issues.

In this chapter we will first take a detailed look on the stock market and the investor relations services as a business environment. After that the problem statement is refined and the requirements of the application are presented. We will also take a look on the actions of the competitors on this field.

2.1 Terminology

In this section the terminology used is defined. The first terms to be defined are related to the securities markets in general, the rest to the investor relations services and the insiders and insider information.

2.1.1 Securities

By securities we mean transferable papers of value, for example shares, options and bonds. For this application we limit ourselves to deal with shares and options since bonds generally don't yield voting power in a company. Thus the ownership structure of bonds is not that interesting. Nowadays most securities are not in paper form, so we can limit the focus on certificateless securities which are in the electronic form.

2.1.1.1 Shares

Shares are securities that give their holders an ownership to a portion of a company. A company may have a multiple series of shares, but all shares from the same series are equal.

A share represents both financial and executive power over the company. Each share (from the same series) is entitled to an equal portion of the wealth and profits of the company. Each share also gives its holder the right to manage the company.

Company's shareholders rarely manage it directly. Instead they use their power by voting in the annual and extraordinary general meetings. The main purpose of those meetings is to elect a board of directors, which in turn chooses the top management and otherwise looks after interests of the shareholders [4].

2.1.1.2 Options

Options don't actually give their holder any rights on the company. They rather promise that their holder may buy company's shares at some fixed price (*exercise price*) in the future (on *maturity date*). If the price of the share is then higher than exercise price the holder may buy shares at exercise price and sell them at market price, thus receiving profit.

Options are often used to motivate and reward the management and other employees of the company. Although they don't actually give any power in the company, information about their ownership is still valuable to the investor.

2.1.2 Public company

A public company is a limited company whose securities are publicly traded in some stock exchange. Public companies usually have responsibilities for publishing information that may have an effect on the price of its securities. These responsibilities may be enacted in the laws of the country, the rules of the stock exchange or in both.

In addition to shares traded in the exchanges public companies may have other series of securities that are not publicly traded. Usually those securities contribute more voting power or have some other privileges. These kinds of arrangements are typical in family-owned companies and are used to ensure

that power "stays in the family". Although these shares are not publicly traded, their ownership structure is an interesting piece of information from the investor's point of view.

2.1.3 Shareholder

A shareholder of a company is a person or an institution that owns shares or other securities. In certificate-less system the shareholder doesn't actually have any material items in his/her possession. Instead the securities he/she owns are recorded in a book-entry account in a book-entry register maintained by a bank or other financial institution.

Information about company's shareholders is (in Finland) publicly available.

2.1.3.1 Nominee registered shareholder

A nominee register is a special kind of book-entry register in which the identity of the actual shareholder is not revealed. Instead, only the name of the institution maintaining the register is publicly disclosed.

Nominee registered shareholders are not able to exercise all rights given by their shares. They are entitled to dividends and directed issues but may not vote at the general meetings of a company.

In Finland, only foreign investors are allowed to nominee register their shares and prevent their ownership on those shares becoming public. They are not obliged to do so and may choose not to in order to participate in general meetings [10].

There has been discussion about the need for such an anonymity since it would be in the domestic investors' interest to have all the shareholders treated equally.

2.1.4 Insider

An insider refers to a person who has some access to some private information that, when published, could have an effect on the price of the company's share. Such information is called *insider information* [25].

Taking advantage of insider information is in most countries forbidden by law as well as its distribution before it is published. According to Finnish law, a person can be given an insider-status either by law (for example members of

the board and auditors are always considered insiders) or by the rules of the company in question. The latter usually applies to some key persons in the top management, sometimes also management secretaries etc. The Securities Markets Act states that the holdings and changes in the holdings by insiders are to be registered and published in the WWW pages of the company in question "without unjustified delay" and kept in registry for five years and in the WWW-site for a year. It's worth noting that in this case, the law specifically states that the information must be available online [30].

The Finnish Central Securities Depository (APK) offers a registry service which automatically tracks ownerships and changes for companies using the certificate-less system. It also acts as a service provider regarding this information, either by providing the raw registry data or by offering companies a possibility to link their web sites to a service called NetSire in order to fulfill the the regulations about publishing registry information in the WWW. The NetSire will be observed in detail in the section 2.2.3.

2.1.4.1 Linkage

In addition to the persons in question, also their spouses, minors under their guardianship and corporations they are considered to control are treated like being insiders, that is securities owned by them are to be registered and published. A corporation in which the insider holds more than half of the voting power is considered to be controlled by that insider [31].

These juridical and natural persons related to the insider are referred as *linkages*. Names of the corporations are public information, names of the persons are not. For example children of the insiders are to be referred as "child of" not by their name [30].

In addition to family members and controlled corporations, the insiders are required to report corporations they have "remarkable influence in", such as corporations in which they act as members of the board. However, holdings of these kinds of corporations are not required to be published [31].

2.1.5 Client

In this thesis client refers to a public company that wants its shareholder and/or insider information to be published in its WWW-site. While publishing shareholder information is mainly an act to please investors, the insider registry is mandatory for each Finnish public company. Companies that fail to comply the regulations are subject to sanctions.

While the company itself is considered to be the client, the users of the the insider registry application consist mainly of individual investors and other parties showing interest to that company. Thus the application must be designed keeping both the client and the users in mind.

2.1.6 Investor Relations Services

Investor Relations (IR) means services a company provides for its owners and creditors in order to help them to better asses the value of prospects of their investment on the company. These services might include historical data on prices of the securities of the company, financial information, estimates on future prospects and information on the ownership structure of the company. Nowadays especially the WWW pages of companies are considered as an important source of such information.

Since especially the up-to-date information on the prices of securities on the stock-market is hard and expensive to get, these services are often outsourced to service providers specialized on this kind of communication. That business is referred as Investor Relations Services.

Outsourced services are often integrated as a part of the WWW site of the company where they should fit seamlessly. This causes a need for customization, services should "look like the client".

It is worth noting that the user and payer of such outsourced services are not the same. Services are bought by the company to be used by its owners free of charge.

Nowadays each and every company has tools for share price tracking available on their WWW site. Hence, the new focus on investor relations is on tools concentrated on the power structure of the shareholders and the activities of the insiders, like a recent competition shows [24]. The aim of this thesis is to generate those tools.

2.2 Objectives

The objective of this thesis is to develop software system for automated acquiring, processing and publishing insider and shareholder registry information. In the process, the suitability of using the Microsoft BizTalk Server in investor relations business is evaluated.

2.2.1 Situation now and the need for change

Currently the insider and ownership registry applications offered are a set of scripts used to convert data received from the APK in *comma separated values* (CSV)-format to XML that is then published in WWW using XSL stylesheets to do the publishing. Data is received by email once a month (and the registry is thus updated once a month) and the publishing process contains a large amount of human labor.

The monthly update of a registry application of a typical client takes about an hour and the process has proven to be quite error-prone (due to the human interaction).

As the amount of clients has been small and registries have only been updated monthly, the current applications have been sufficient. The new legislation, which was enacted on July 2005, however demands the information to be updated "without unjustified delay" which can be interpreted as "daily". Also after the legislation the number of clients is predicted to increase heavily, since having such a register on the WWW-site becomes mandatory. The law gives public companies a transfer period of maximum one year, after which the register must be published on their WWW-site [31]. Thus a large number of new clients can be expected before July 2006.

The existing application also used data in a format that won't be available from APK any longer, since they are replacing CSV with XML, thus there would be a need for refactoring to accept the new data. In addition to that the existing application could not display all the data required by the new legislation (mainly the individual trades and the names and holdings of the linkages).

In addition to changing the format of the data, APK will also be offering a new method for transporting the data, after their new extranet service will be opened. Thus instead of getting it via email, it is possible to fetch data using HTTP protocol. Data will be cheaper through the extranet and if the fetching process could be automated, need for human interaction would disappear completely.

For the reasons given above, the registry application must be redesigned and re-engineered. Having tens of clients each requiring an hour of human labor each day is not possible or affordable. Also since the information will be a mandatory part of the web sites of the clients, and it will be required to be up to date, a manual process would generate a need for backup personnel in case of illness or other absence.

2.2.2 Criteria for a successful solution

Since the new legislation demands the insider register to process and display data that the old application is not prepared for, enabling processing of that data can be considered as the most important requirement. In addition to that, there are two other requirements worth highlighting here. They are presented next.

2.2.2.1 Minimal need for human interaction

The main reason for this software project is the time-consuming nature of the monthly updates of the application currently in use. A typical client installation takes about one hour to update every month. After the new legislation, updates should be done weekly or even daily. Thus the current situation is unbearable.

Minimizing human interaction also removes the risk for human errors from the process. Since especially insider information is required by the law to be accurate, errors are not acceptable.

A successful solution would remove the "update process" completely and instead transform the insider and ownership registry applications into long running instances that update themselves from some external data source, much like the share price tracking applications have already been implemented.

2.2.2.2 Easy addition of the new data sources

The new application is directed only to the Finnish clients, but since the new legislation about publishing insider information is based on a directive of the EU [30], it will be effective union-wide. Therefore it would have potential markets elsewhere in the Union. The raw data needed for this application will most likely come in different formats from different countries, but it would be desirable to be able to add new sources of the data to the system with minimal effort.

2.2.3 Previous and competitive implementations

The situation of the market is interesting, since all competitive implementations must utilize APK's data due to its nature as a registry holder by law. The fact that APK also offers outsourced insider register services on the WWW makes the situation even more interesting. By the writing of this

thesis, there are no services in Finland offered by any other organization than APK.

APK's application is called NetSire [28]. It offers all the information required by the law, but its appearance is not customizable, hence it can not be embedded into the web-page of the company. Additionally, its user interface has been criticized [24]. Nevertheless, it has the advantage of being directly linked to the data, without somewhat unusable XML interface in between.

Other competitive implementations include those made by companies themselves (or by the provider of their WWW site). Those solutions probably will not endure the new requirements, for example the daily updating cycle.

Chapter 3

E-Business

Within this chapter we shall cover the concept of e-Business or e-Commerce. It can be defined as broadly as "activities conducted using electronic data transmission via the Internet and the World Wide Web" [5]. If interpreted loosely, the definition can be seen to classify almost any business activity conducted in the modern world.

Even though publishing insider information in the Internet for an investor to examine can be seen as e-Business, within this thesis we are mostly interested in the subsection of electronic commerce that is conducted without intervention from human users. This kind of activity can be called the Electronic Data Interchange (EDI). The term may also be used in a more limited context, referring to a standard of electronic communications called the Electronic Data Interchange For Administration, Commerce and Transport (EDIFACT). Within this thesis we use the formed interpretation which will be elaborated more in the next chapter. After that we will see how the task covered by this thesis can be seen as an e-Business implementation [8].

3.1 Electronic Data Interchange

3.1.1 The definition

The Electronic Data Interchange (EDI) can be defined for example as follows:

> *EDI is exchanging data in electronic form between two organizations or information systems via data medium or data line using messages with a defined structure [23].*

The definition above summarizes most aspects of the electronic data interchange. They can be further elaborated as follows.

EDI is data exchange when both parties are non-human. Thus EDI messages are not intended to be *human readable.* They are rather in a form that both information systems understand. For example email messages or web pages do not fall into the this category [23].

This data exchange is done electronically, via some data medium. Exchanging information on paper should not be considered EDI.

The definition does not cover the nature of the data being transmitted. It may contain for example invoices, orders and order confirmations. But aside from administrative documents there may also be catalogs, technical data sheets and other information to be exchanged.

Messages have some (well) defined structure they comply. The need for defined structure arises from the fact that both ends of communication are not humans, thus the messages must be unambiguous [23].

3.1.2 History

Information systems based on computers have for a while been used in many companies. Such a system replaced much manual filing, planning and calculating work and could contain almost all the data about company's operations.

But such systems had their boundaries. When dealing with customers on the one hand and with suppliers on the other, companies needed to deal with people. Data needed to be extracted, by a human, from one company's system in the form of an order, a quotation or an invoice and then sent via fax, mail or telephone to the other company. Usually it was then inserted, by a human, to the other company's information system. This required a lot of labor and was of course error-prone. In the figure 3.1 we see the flows of information between those two information systems. A question soon arose: Could information systems communicate themselves, without human aid or intervention [5]?

3.1.3 Three forms of EDI

The definition of the Electronic Data Interchange only requires the information systems to exchange data electronically via some kind of data medium,

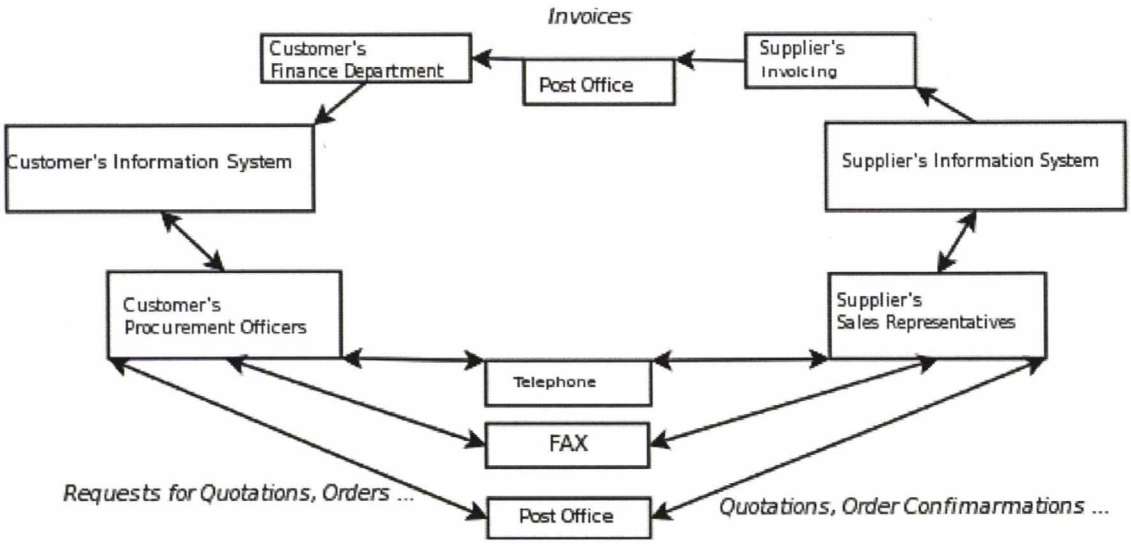

Figure 3.1: Exchange of information between two companies before implementing an EDI-system [5]

it does not dictate the exact medium to use. There is thus several ways to conduct this exchange, they are illustrated on figure 3.2.

3.1.3.1 Direct Connection

This form of communication requires a dedicated line from one system to the other. That line can be anything from a leased phone line to a simply dial-in modem line. All data transmission is done via that line, directly from one information system to the other. The first implemented EDI systems used direct connections [5].

EDI using a direct connection is physically easy to implement, the two systems are just plugged together.

The problem with this approach was its high costs of implementation. Leasing phone lines was expensive and each additional partner caused a need for an additional line. Thus companies were usually not that eager to establish EDI communications with smaller partners contributing only a couple of transactions in a year. Due to this, companies needed to maintain ability to communicate with some partners in the traditional way. Maintaining multiple systems on the other hand increased costs [5].

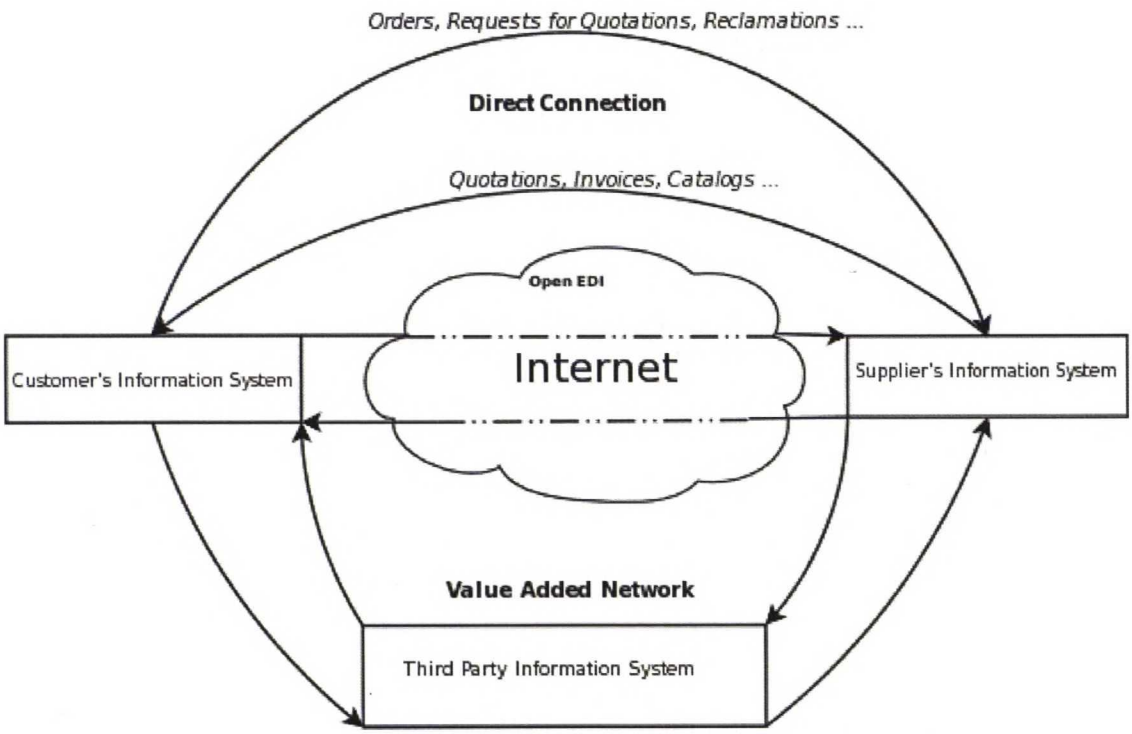

Figure 3.2: Three possible ways to exchange information using EDI [5]

3.1.3.2 Value Added Network

Many small firms cannot afford the high implementation costs of direct connection EDI system. For such business partners, the Value Added Network was an affordable solution. Such services were offered by some third party firms [20].

The Value Added Network is essentially a collection of incoming and outgoing mailboxes for each user of the network. The network itself is responsible for picking up messages from outgoing mailboxes and delivering them to the incoming message-box of the receiver. Messages are stored in the mailboxes, so there is no need to be connected to the network all the time. Instead, message-boxes can be checked for new messages periodically [20].

In addition to savings Value Added Networks also provide their users tracking and auditing services. It can for example acknowledge to the sender that the receiver has received the message or keep a transaction log of the messages exchanged [8].

3.1.3.3 Open EDI and the Internet

The dawn of the Open EDI begun when expensive leased lines and dial-up lines were replaced by the Internet connections. In the Open EDI system documents are transmitted from one system to another using the global Internet and protocols related to that network. Open EDI is a cheap solution, since most firms already have an Internet connection available. The questions hindering the expansion of Open EDI mostly are those related to confidentiality and security. Since business documents travel in the Internet along with other network traffic special consideration must be applied to their integrity and safety [20].

3.2 Insider register as an e-Business solution

As stated in the beginning of this chapter, from the point of view of the e-Business, the insider register can be divided into two parts, the one acquiring and processing the registry information and the other publishing it. The latter can easily be seen as a form of e-Business, even though the earnings logic is somewhat odd: the entity using the service to find information for example for evaluating an investment decision is not the same entity paying for the service. As has been seen in the section 2.1.6, this is usually the case

with investor relations services

In the section 3.1.1 EDI was defined as electronic exchange of business documents between information systems via some data medium. The former part of the insider register indeed does that, although the exchange is a bit one-sided, since the role of the insider register is mainly to receive documents from the APK's system. As we shall later see, the exchange is actually not so much done between information systems, rather it needs a human intermediator. In that light, the insider register cannot be considered as an EDI system at all.

There is also another difference. When considering EDI systems, the exchange of documents is often seen as a support process for the actual business. For example, quotations are exchanged in order to buy some items or services. In this case, the documents and the information in them are the business, not only supporting it. But that is often the case with the modern information industry.

Chapter 4

Extensible Markup Language

Nowadays the *Extensible Markup Language* (XML) [35] has become the standard way of representing and storing information. In this chapter we shall take a look on the history and definition of the language as well as examine some XML based languages that are utilized in this thesis: XSD and XSLT.

4.1 Introduction

Although XML stands form Extensible Markup Language, it itself is not a language but a tool for defining new languages. Thus we can speak about "XML based languages" or the "XML family" when referring to those languages that are defined using XML [21].

XML is derived from *Standard Generalized Markup Language* (SGML), which was also a definition tool, not a language itself [21]. XML is still compatible with SGML [35]. The specification of XML is managed by World Wide Web Consortium (W3C).

XML is a *textual format*, which basically means that it is readable and writable by any text editor, without the need for a dedicated software (even though there are ones). There are also numerous pieces of software for parsing and processing XML. Thus XML is a format, that is easy for both humans and computers to read and use [21].

An *XML document* is defined to be a piece of data (for example a file in a file-system) that is *well-formed* [35]. Next we will examine the content of the document and after that the rules that make a document well-formed.

4.1.1 Parts of the XML document

A document must begin with a *prolog*, which specifies the version of the XML that is being used [35]. Prolog may also state the character encoding used. In addition to the XML declaration, prolog may contain a *document type declaration*, which defines the *grammar* the document uses. The ways to define grammars are discussed later, on section 4.2.

After prolog comes *text*, which in XML can be either *character data* or *markup*, character data being everything that is not specifically markup. There are multiple different types of markup, the most commonly needed being tags, comments and processing instructions [35].

Tags may be either *start-tags*, *end-tags* or *empty-element tags*. A tag is delimited by '<' and '>' characters. A tag contains the name of the tag and if it is a start tag, it may contain any number of *attribute definitions*, which are pairs of the name of the attribute and its value. Unlike with HTML, each attribute must have a value. For example, the following is a start-tag: <book price="12.2">. [35] An end-tag may not contain any attribute definition and the name of the tag is preceded by '/'. An end-tag is said to *close* a start-tag with corresponding name. For example, the start-tag above would be closed by </book>. XML is case sensitive, so </BOOK> would not close it [35].

An *element* is delimited by start-tag and corresponding end-tag and it contains everything between them, including the tags. An element which contains only start- and end-tags, and no other element or character data in between, is said to be an *empty element*, and can be abbreviated using an empty-element tag, which is a start-tag with a '/' in the end. For example: <book price="12.2"></book> could be expressed more shortly by <book price="12.2"/>. Empty-element tags exist only to save some space, logically they are equivalent with the longer form [35].

Comments are a set of characters, contained between '<!–' and '–>'. They are treated like comments in programming languages, so XML processors completely disregard everything contained by a comment [35].

Processing instructions are much like tags, but they are delimited by '<?' and '?>', and there are not start- or end-instructions. Processing instructions are needed in order to invoke some external applications while processing the XML document. They exist for the sake of compatibility with SGML and their usage is not encouraged [21].

4.1.2 Well-formedness and tree structure

A well-formed document must contain at least one element. Thus an empty document (or a document containing only a prolog) is not a well-formed one. Each document has exactly one element that is called a *root-element* or a *top-element*, which is not contained by any other element [35].

All other elements must be properly *nested*, so the start- and end-tags of an element must both be contained by the same element. For each element in the document that is not a root element, there must be another element containing it. That element is called its *parent* while the element itself it referred as *child* of its parent [35].

Due to the nesting and requirement for a single root-element, each well-formed XML document can be considered as a tree, which has its root-element as a root. This is the way that many XML parsers use to treat those documents.

4.1.3 Namespaces

In natural languages, the meanings of words are not unique. For example, chair may refer to an actual piece of furniture or a person acting as a chairman of the board. Human users of the language are (usually) context aware being able to not to sit on a chairman, but applications do not have such luxury.

It is often common to reuse earlier defined sets of names and meanings of the elements in multiple different types of documents. These sets are called *vocabularies* and will be defined in section 4.2 in more detail. Reusing saves resources, since there is no need to define same vocabularies over and over again, but can also cause problems, since multiple vocabularies may have different meanings for the same name of the element [36]. Using such vocabularies in the same document leads to context problems.

XML has a way to link elements into certain context. *Namespaces* [36] are used to distinguish elements with a same name from one another. Using namespaces still does not give applications context awareness, but it allows users to state that a chair located in a conference room is a different kind of element than a chairman sitting on that chair.

Each namespace is identified by *Internationalized Resource Identifier* (IRI) [9], which is basically an extension of *Uniform Resource Identifier* (URI) [2] that allows a larger set of characters to be used. Most commonly (in namespaces) used types of IRIs are perhaps *Uniform Resource Locators* (URL), for

example `http://www.w3c.org/index.html` [36].

Although IRIs are resource identifiers or locators, the IRI used for namespace identification is not required to refer to any resource. Basically namespace identifier is just a string with certain syntactic requirements [32].

After introduction of namespaces, both name of the element or attribute and its namespace can be used to define in which vocabulary it belongs to [36]. Since announcing possibly long IRIs in addition of element or attribute names, XML allows using *prefixed names* which are combinations of a *prefix* and a *local name*, separated by a colon. The prefix, which can for example be an acronym, has earlier in the document been bound to a certain IRI and should only be considered as a placeholder for the IRI it refers to [36].

4.2 Defining languages - DTD and XSD

XML is not a language itself, it only offers a way to define languages. The elements and attributes in a XML-document have no meaning by themselves. In order to use XML efficiently in the electronic data interchange between enterprises, we need to define the vocabulary and grammar we are using while communicating.

A *grammar* describes the structure of the document. It defines the children and attributes an element can have and the data-types they may contain. But a grammar cannot describe the meanings of those elements. For that purpose there are *vocabularies*, which give the data that the document contains the actual meaning. Vocabularies can be either official or informal, but they are all defined in some natural language, since there is not yet a language that can be used for that purpose. When establishing business relationship with another partner, usually both grammar and vocabulary of the language used are needed. Since there are no formal languages for vocabulary definition, we won't discuss it any further [11].

When a grammar has been defined we can introduce a concept of *validity*. As stated before, each XML-document must be well-formed, that is to comply the general syntax rules of XML. But the document can also be valid. The validity exists only against certain grammar and a valid document complies the rules of that grammar.

The XML-family has two members that can be used to define grammars and thus new languages. The earlier language is called *Document Type Definition* (DTD) and it is originally used with SGML and thus it is not a XML-language

at all. The *XML Schema Definition* (XSD) is younger language, defined for the XML. The latter one is also an XML-based language itself.

Document conforming some grammar (defined either using DTD or XSD) is said to be *valid* (against that grammar). Validity is a stronger concept than well-formedness defined previously, since each document must be well-formed (otherwise it's not an XML-document), but a well-formed document does not have to conform grammar it declares, or it may even choose not to refer to any grammar at all and still be well-formed [11].

4.2.1 DTD

As has already been discussed, DTD is the older one of the two languages used to define grammars. It originates from the time of SGML. The notation used when writing DTDs is called *Extended Backus-Naur Form* (EBNF) and it differs significantly from the XML [22].

DTD allows definition of both elements and attributes. Definition of an element is unique in a sense that the definition must be the same throughout the document. Thus the element cannot be defined differently with regards to its parents or any other information related to its location in the document tree.

An element may be defined to have *empty*, *any*, *mixed* or *element* content. The first two are defined by a specific keyword in the elements definition, the last ones with simply giving the spec of the content. Element defined to contain character data is always considered to be of mixed content, even if the grammar does not allow any sub-elements. The character data may not be given any further spec, for example it cannot be stated that some element must contain data that can be treated like a number of date. Mixed content is just mixed content, without any further classifications [11].

For the mixed and element contents the sub-elements of the element must be stated. DTD allows definition of both *sequences* and *choice groups* of elements (or character data). These may be nested to arbitrary depths. Each sub-element can also be given an *occurrence indicator*: optional, optional and repeatable or required and repeatable. The last two state that there may be more than one occurrence of that element (in a sequence). If the indicator is omitted the element is considered to be required and not repeatable. Occurrence indicators can not state the exact number of occurrences, but just whether or not there may be more than one element of the same type. Thus there is no easy way to define for example that a <vehicle> must have two or four <wheel>-sub-elements. DTD is thus quite limited in defining complex

grammars [11].

DTD has also tools for defining attributes. An element may be declared to have an *attribute list*, containing all the attributes the element has. For each attribute, its name, data-type and default value. Attribute may be defined as either mandatory or optional and it may even be given a mandatory value. Tools for defining the data-type are as limited as they are with the elements. In addition to plain character data, only data-types allowed are enumerations, tokens, which are basically just pieces of character data with limited allowed set of characters and ids and references to ids. The last two can be used to define attributes serving as unique identifiers or references to unique identifiers. DTD does not allow restricting the values of the attributes on any other way, for example requiring some attribute to be a number or a date is not possible [11].

4.2.2 XSD

XML Schema Definition (XSD) is a new and improved approach in defining grammars. A grammar is defined in XSD document which itself is a well-formed XML document. Feature-wise XSD is backward compatible with DTD, so there are actually no reasons for using DTD instead of XSD [3].

Compared to DTD, XSD allows more precise data-type definitions. Instead of plain character data, element or attribute may be defined to contain integers, floats, dates and so on. In addition to pre-defined data types, XSD allows definition of new types. These new *simple types* can be derived from previous ones either by *restriction* or *extension*, that is either giving a stricter guidelines the content must follow or by allowing the content more freedom. The content of an element may thus be stated to be a zip-code, a social security number etc. [3].

XSD also allows more flexibility in defining element content. Like with DTD, sequences and choice groups of elements may be defined. In addition to that, XSD offers an *all*-group, which states that each element mentioned in that group must exist in the document, but the order does not matter. That kind of modeling cannot be done with DTD [3].

XSD also enhances element content definition by allowing elements to minimum and maximum occurrences, which can be either integer numbers or a token *unbounded*. Thus using XSD it is possible to define exact amount of elements required (instead of default one). In addition to single elements, also groups may have occurrences, so that for example choice-group may be declared optional or repeating two to three times, for example [3].

Schemas defined by XSD can also contain special kinds of comments, that, unlike the normal XML comments, are usable by both human and applications. They are placed inside *annotation* element, which can be contained by most of the building blocks of the schema [37]. The contents of the annotation element do not affect the validation process using that schema, but since they are not actual XML comments, they are part of the document tree and processors are not allowed to discard them. We shall see one use of the annotation element in section 5.5.1.

4.3 Transformation and publishing - XSL and XPath

The XML language was developed in order to separate content from its presentation. The content described by an XML document can be desired to be presented in multiple ways, for example with a web browser or in printed format.

The Extensible Stylesheet Language (XSL) has been evolved from the Document Style Semantics and Specification Language (DSSSL) used for publishing SGML (much like XML itself has evolved from SGML). Originally XSL was used to publish XML documents in a printed form, the publishing was a two-phased process: at first the elements from the source document are selected, reordered and combined (this is called structural transformation), after that they are formatted for publishing. It soon appeared that those two phases needed two separate languages, thus XSL was divided into *Extensible Stylesheet Language: Transformations* (XSLT) and *Extensible Stylesheet Language - Formatting Objects* (XSL-FO). From the original publishing process became a process of using XSLT to transform a source document into XSL-FO. XSL-FO is essentially just another XML language, and it is not covered any further in this thesis. The XSLT on the other hand can be used to transform XML document, not only to XSL-FO but to any other XML language (as well as to HTML and to any text-based language) [12].

This section deals with the XSLT language as well as the XPath language used by the former for pattern matching and some other tasks.

4.3.1 Characteristics of XSLT

XSLT is an XML language used to describe transformation processes. Its syntax follows all the rules of an XML document, and its namespace is `http://www.w3.org/1999/XSL/Transform`. Due the fact that XSLT is and XML language used to process documents written in XML, XSLT documents can be used as inputs of XSLT transformations [12].

In addition to being an XML language XSLT is a declarative and purely functional language. With declarative we mean that programming with XSLT is describing *what* the program shall do, not *how* it shall do it. This differers greatly from the usual procedural approach. When using declarative languages, we are more interested in the outcome than the process [12].

Functional programming languages on the other hand contain set of functions that take arguments and return results. Those functions can not cause any *side-effects*, that is they are not allowed to alter or modify either their arguments or the global environment. Thus a function called with a same arguments always gives the same result (if run at the same environment). This makes the order in which the functions are executed irrelevant [1].

Functionality and declarativeness together make an XSLT document a set of rules (called *templates*) that are not in any particular order. We shall examine the templates more in detail later. The document itself can also be called a *stylesheet* [34].

4.3.2 Usages of XSLT

Nowadays usages of XSLT can be divided into two separate categories: conversion and publishing. Next will take a look on those two usage separately.

Conversion is essentially converting a document written in one language into another. The need for this process can arise for example from the need to integrate two applications operating on two different XML vocabularies (and thus two different XML schemas). The conversion is needed in order to allow these two to understand one another.

But as wide-spread the XML is today, there are still applications and systems operating on languages that are not XML based (CSV, some EDI dialects). XSLT can handle these situations as well, converting XML to plain-text. It is even possible to use XSLT to convert plain-text to XML (or better yet plain-text to plain-text). This can be achieved since XSLT does not actually operate on an XML document but a tree presentation of such a document,

as we will shall later see. Thus in order to transform for example CSV using XSLT we only need a parser to create a tree presentation of the CSV document [12].

The difference between publishing and conversion is that conversion produces data for another application to use and read while publishing targets to human beings. But essentially publishing is conversion where target language is something that a human being should be able to understand. These two usages are separated here only in order to emphasis the versatility of XSLT [12].

4.3.3 Transformation process

The XSLT transformation process operates on a *source tree* and produces a *result tree* as its result [34]. Thus the transformation is applied on the tree presentation of the document, not on the document itself. Those two trees are separate, thus their structures can be completely different [34].

The transformation process is directed by one or more templates (which are discussed further in the next section). The transformation process begins from the root of the source tree. The correct template is searched for and then instantiated. If the template contains references to another templates, also they are instantiated [34].

XSLT also contains some build-in templates [34], that are automatically effective. Most of them are simply passing their input through into the output. Thus it is for example quite easy to write a stylesheet that simply transforms one kinds of elements into something else and leaves the rest of the document untouched.

4.3.4 Templates

A *template* is a basic unit of an XSLT document. Each template describes a transformation process performed to process some output. A template can be invoked by rule, by name or both. Templates invoked by name called *named templates* and they are very much like functions or subroutines in every other programming languages [12].

Templates called by the rule on the other hand are defined by a *pattern* which describes (using XPath) what kind of nodes in the source document the template is supposed to be applied. Templates called by the rule are applied to every node that matches the pattern. Formally, the concept of

pattern and matching can be defined as follows:

> *The Node N matches pattern P if and only if there is a node A that is an ancestor-or-self of N, such that evaluating P as an expression with A as a context node returns a node-set that contains N [12].*

The concept of evaluating XPath expressions and the results of such evaluation are discussed in the next section. Usually, the patterns used in an XSLT document are fairly simple, for example matching just on the name of the element.

The evaluation of a template is called *instantiation* and it is initiated when a node is being processed and the template is the best template to match that node. As a result, a *result tree fragment* that can be inserted into the result tree is produced. The fragment can contain multiple elements, but it still is not treated as a node-set. This hinders development of templates that used the outputs of other templates as inputs, since only the string values of outputs can be used [34].

A template can contain plain text or XML markup (all XML elements that do not belong into XSLT namespace are copied into the result tree fragment) and also XPath expressions to create elements or text [34]. XSLT contains some elements that are used within a template to guide the production of the result. These elements are available for example for conditional processing or looping [34]. A template can also call or refer to another templates.

4.3.5 XPath

Like XSLT also the language XPath operates on tree representation of the XML document in question. XPath is a separate language, designed for referring to parts of an XML documents. It is mainly used within XSLT for pattern matching. In addition to addressing, XPath also contains certain necessary functions for number and string manipulation as well as boolean algebra [33].

Unlike most of other members of the XML language family, XPath does not use the XML syntax. This enables for using XPath expressions as attributes of XML elements [33].

As previously said, XPath operates on the tree representation of an XML document. As seen by XPath, such tree contains following kinds of *nodes* [33]:

- root node
- element node
- text node
- attribute node
- namespace node
- processing instruction node
- comment node

Like an XML document only has one root element (called document element), also an XPath tree has only one root node. But this node does not correspond the document element, rather the document element is an element node that is a child of the root node [33]. For each element in the XML document there is an corresponding element node in the tree. Text nodes on the other hand represent the character data in the document.

Each node can be represented by its *string-value*, a way to express the node as a single string. For text nodes it is simply the character data itself. The string-values of other types of nodes are a little bit more complicated to construct, for example a string-value of a element node is a concatenation of the string values of all its children [33].

The core of the XPath are *expressions*, which produce, when evaluated, either an object of a basic type (boolean, number or string) or a *node-set*, that is an unordered collection of XPath nodes [33]. Those expressions are evaluated in a *context*, which contains most of all the *context node*, the current "root" of the evaluation [33].

4.3.5.1 Location Path

The most commonly used type of expression in XPath is a *location path.* A location path can be either absolute or relative. Absolute paths are traveled from the root node, relative ones from the current context node [33].

A path is a sequence of *steps.* A step is composed from *axis* and *node-test* and in addition contain *predicates.* The axis defines the direction in the tree of the step (upwards, downwards, sideways and so on) and the node-test is usually simple a name of the target of the step, most commonly the name of the element. Predicates can be used to refine the step even further [33].

Each step returns a node-set that contains all the nodes in the tree that have the correct relationship to the context node and conform the node-test and the predicates [33]. The return value of a location path expression as a whole is a node-set containing all the nodes that are selected by each of the steps.

Chapter 5

Microsoft BizTalk Server

5.1 Introduction

Microsoft BizTalk Server is a tool for business integration. It is designed to support electronic data interchange both within and between different organizations. BizTalk is essentially a message router and converter, used to enable communication, but it can also be seen and used as a software development platform.

When used within an organization, BizTalk's role is to receive messages from one information system, do the necessary conversion and send them to appropriate receivers. This enables using several different kinds of systems, that do not necessarily understand each other's messages, to do various tasks. BizTalk could for example route messages from *Enterprise Resource Planning* (ERP) system into purchasing application and backwards. This kind of usage is called *Enterprise Application Integration* (EAI) [15]. An example of such a setup is presented in the figure 5.1.

In communication between different organizations, like illustrated in the figure 5.2, BizTalk can be used to receive and send messages to and from information systems of the clients and suppliers of the company, no matter what kind of integration platform (BizTalk or some other) they use for messaging. Usually this kind of communication is conducted via the Internet. BizTalk could for example receive purchase orders from customers via a web service and order components from a supplier via email. This kind of activity is referred as *Business To Business* (B2B) integration [15].

Or course it is possible to combine techniques presented above and use BizTalk in both internal and external messaging.

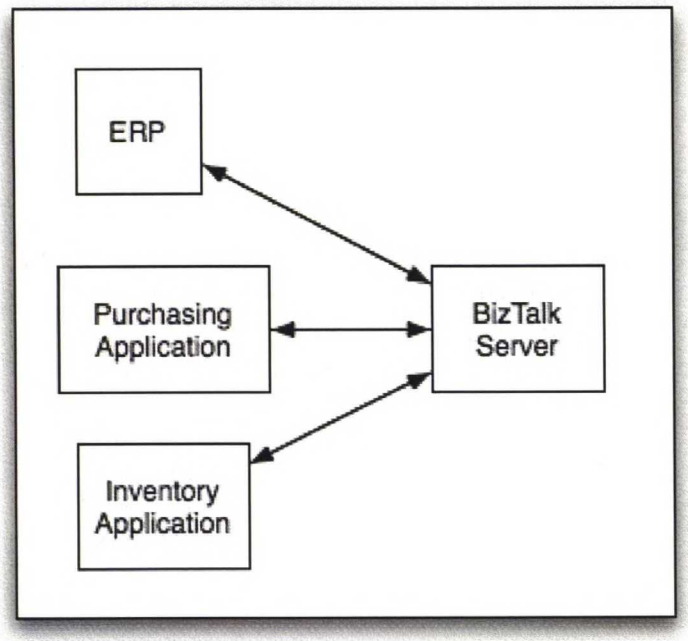

Figure 5.1: BizTalk used in Enterprise Application Integration [15]

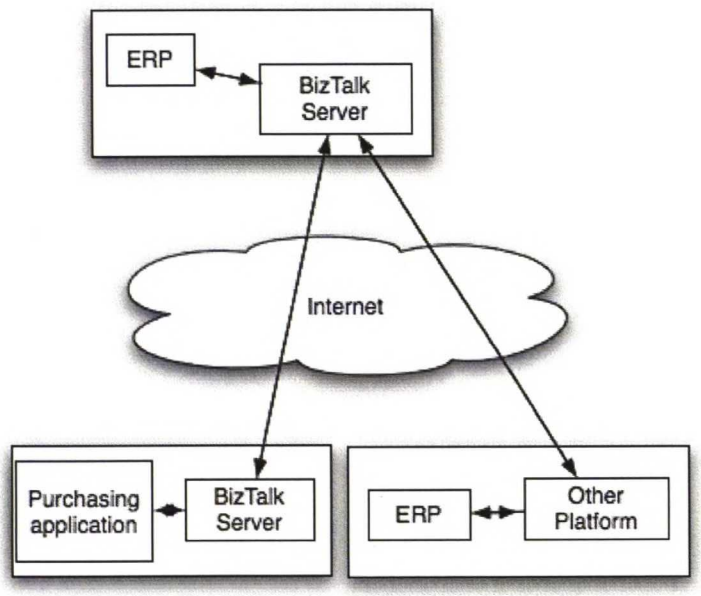

Figure 5.2: BizTalk used in Business To Business Integration [15]

This thesis uses the term BizTalk Server to refer to the BizTalk Server 2004. There some are major differences between version 2004 and previous versions 2002 and 2000. Those differences include not only different naming conventions but also some differences in the functionality. Thus contents of this chapter are not applicable to BizTalk Server 2002 and 2000.

BizTalk Server is a part of Microsoft Server System. It depends quite heavily on other members of this family, using for example Microsoft SQL Server as a database engine and Microsoft Visual Studio as a developing environment [13].

BizTalk, like other members of the Microsoft Server System, is also a part of Microsoft .NET. In the following section we will take a look on .NET Framework which is the tool that actually builds and runs the applications developed for BizTalk.

5.2 .NET Framework

The .NET Framework is a component integrated heavily on the Windows operating system itself. It both builds and runs applications that are used under Windows.

The .NET Framework contains a set of class libraries for both standard input/output operations and interacting with various more complicated interfaces such as databases and Web Services using XML.

The most important component of the .NET Framework is the Common Language Runtime (CLR). That component is responsible for the actual running of applications. Applications run under .NET Framework are first written in some of the various languages supported by the Framework such as Visual Basic or C#. After that they are compiled to an intermediate language, Microsoft Intermediate Language (MSIL), which is turn is interpreted and run in CLR. A set of such code is called an *assembly* [16]. The intermediate language is independent of machine architecture or processor family. The concept is very much like the one used in Java with it's Virtual Machine, although .NET Framework is not a complete machine in that sense [13].

Although the specifications of the CRL and .NET's class libraries are publicly available, the only usable implementation of those so far is maintained and owned by Microsoft[1]. Thus using the .NET Framework also dictates the

[1] Although there are some open source projects implementing .NET interfaces, for example, `http://www.mono-project.com/`

operating system and server architecture used to run the applications. This may be considered as a problem, since .NET Framework is thus not machine architecture independent in the same sense as Java [13].

A machine with CLR installed also has a code cache called *Global Assembly Cache* (GAC). GAC is a specific folder in the file-system in which all globally accessible and referrable assemblies are stored [16]. BizTalk assemblies differ from normal .NET assemblies because in addition to being installed into GAC they will also have to be registered in the BizTalk Configuration Database. This process is called *deployment* [17].

5.3 Software development with BizTalk

Usually software systems are designed and developed by professionals who have IT background. Development is done using some programming language that needs specific training and experience to be fully utilized. The role of the actual users of the system, those who know the application field best, is minimal: they may take part in designing, but not in development.

BizTalk takes a different approach. The main idea is to minimize need for software development trained personnel in a project and give as much tasks as possible to persons familiar with the application field. This is accomplished via three different roles used to create and maintain BizTalk applications. These roles and their responsibilities are described next.

The *business analyst* defines the process on hand using high level tools to describe flows of information, different types of business documents and mappings between them [6].

The role of the *developer* is to implement processes defined by business analyst. This includes constructing the XML schemas, implementing orchestrations and mappings between documents in detail [6].

The third role is that of the *administrator*, whose task is to deploy the applications and maintain and monitor their health [6].

5.4 Message processing overview

Routing and processing messages is the key functionality of a BizTalk Server. Next we will take a look on how the messages are processed within a BizTalk server and trace their path through the system. The flow of a message can

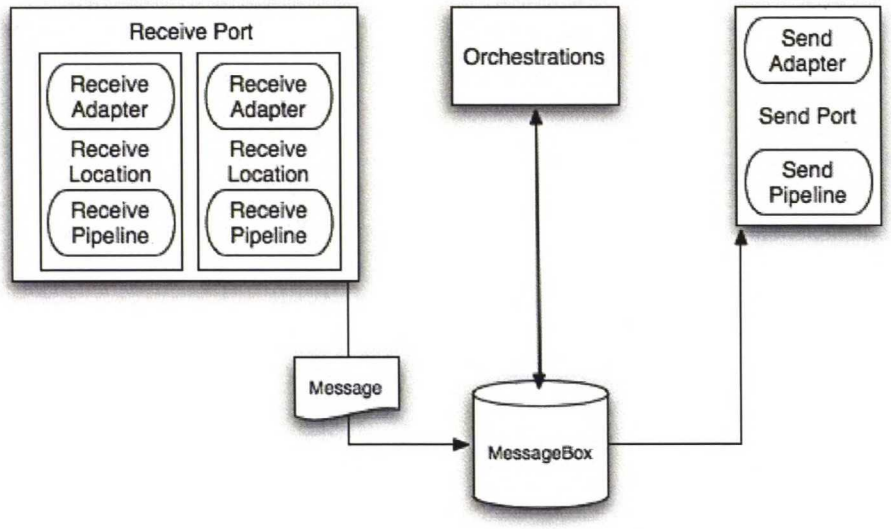

Figure 5.3: The flow of a message within BizTalk Server [15]

be followed from the figure 5.3 [15].

It is important to note that once a message is created, it is immutable [19]. Thus, no changes can be applied to message, instead a new message must be created. The need for immutability rises from the fact that BizTalk's messaging engine may deliver one message to several parties at the same time. Thus concurrency problems might occur if one or all of them were allowed to modify the message.

5.4.1 Receive Port

Incoming messages enter the system via Receive Ports. Each port consists of one or more Receive Locations, which in turn are a combination of an Receive Adapter and Receive Pipeline [19]. Next we will examine them in detail.

5.4.1.1 Receive Adapter

Adapters are the real point of entry to the BizTalk-system. An adapter is an implementation of a certain method of communication, for example a protocol. All adapters a derived from a framework called Adapter Framework [6].

Adapter	Purpose
SOAP adapter	Communicating with WebServices using SOAP-protocol over HTTP
BizTalk Message Queuing adapter	Communicating with other BizTalk Servers
File adapter	Receiving messages directly from the filesystem, for example from one directory
HTTP adapter	Fetching data using HTTP-protocol or exposing an URL for other applications to connect
SMTP adapter	Receiving messages via email
SQL adapter	Fetching information from a SQL Server database
Base EDI adapter	Receiving messages using standard EDI-protocols (X-12 and EDIFACT)
FTP adapter	Receiving messages using FTP-protocol

Table 5.1: Adapters shipped with BizTalk Server [6]

Since the purpose of a BizTalk Server is to integrate different kinds of systems communicating in different ways, the adapter library shipped with the Server is substantial. In addition to adapters provided by Microsoft a developer is free to use any adapter provided by a third party or to develop one himself. The table 5.1 lists all adapters provided with BizTalk Server by Microsoft.

5.4.1.2 Receive pipeline

After the message has entered the system via an adapter, it goes through a receive pipeline. Although BizTalk Server can receive messages in many formats, the internal format used by the server is XML. Thus the incoming messages must be converted into XML documents. This task is called disassembling and it is done in the receive pipeline [6].

After disassembling the pipeline can perform tasks related to the authenticity of the incoming message such as validating it against a schema or verifying its digital signature [6].

There are some predefined pipelines shipped with BizTalk Server, but a developer can also create custom pipelines using tools included in the BizTalk [6].

5.4.2 Message Box

After coming through a receive port messages are stored in a message box implemented as a database on Microsoft SQL Server. This process is called *publishing* the message. From the database the message is delivered to all parties that have *subscribed* it. These subscriptions are handled by *messaging agent* and can be based on both content and context of the message. Subscribing parties can be either orchestrations or send ports, both of which will be discussed in detail later [19].

Usually messages are first subscribed by an orchestration, which in turn may create new messages and publish them into the message box. These messages may in turn be subscribed by a send port.

In addition to bodies of messages, BizTalk Server uses the database as a persistent store for states of the orchestrations, message queues and parameters of its operations [19].

5.4.3 Send Port

As the BizTalk is usually used to transform and route messages, it is natural that some messages will eventually be sent out of the system. This is done via Send Ports and Send Port Groups.

The structure of a Send Port is much alike the one of a Receive Port. The main difference is that there are no "send locations" but each port is itself a combination of a pipeline and an adapter. All adapters available to the Receive Locations are also available to Send Ports (listed in the table 5.1) [6].

Sending the same message into multiple destinations is done using a Send Port Group which is essentially only a collection of Send Ports, all sending the same message into different destinations [6].

Send ports subscribe messages either by filters defined as properties of the port or when they are bound to logical ports of an orchestration. Essentially both ways to subscribe are handled in the same way.

5.5 Key elements of BizTalk

The core idea behind the BizTalk are messages and their processing. BizTalk provides multiple tools for describing these processes. The ones used most often are schemas, maps and orchestrations. These are also the tools that the developer spends most of the time with. Pipelines, adapters and other supporting components can usually be taken "out of the box" and used as is, but schemas, maps and orchestrations must be designed to be best suited for the problem at hand.

5.5.1 Schemas

Schemas are used to describe structure of the incoming, outgoing and internal messages. The version 2004 of BizTalk Server uses XSD (XML Schema Definition) language in schema description, the structures of messages are expressed with XSD documents [6]. Earlier versions used a language called *XML Data Reduced* (XDR) for this job. Since XSD is a standard and XDR was a language defined by Microsoft and other, this can be considered an improvement. Additionally, the expressive power of XDR is weaker that that of XSD, even though it support more complicated structures than DTD [14].

Schemas can be created writing XSD by hand, but this is often a difficult task if data-structures to be defined are complicated. Therefor BizTalk provides user with a graphical tool called BizTalk Editor, which can be used to design schemas. Existing schemas can also be imported to a BizTalk application. Schemas used like this are usual provided by integration partners to describe message structures they are using for communication. If a partner fails to present a formal schema for the messages, BizTalk contains also a wizard to create schema using a document instance as a basis [6].

The XSD language can only be used to describe schemas of the XML messages. BizTalk can process other kinds of messages also, for example fixed length and delimited text messages. Many older enterprise applications still communicate via such messages, so supporting them is essential. For describing their structure, BizTalk uses so called *flat file schemas*. They are normal XSD documents which use XSD's *annotate* element to store the definitions of the flat file positions [17]. This is a bit un-orthodox use of the XSD but the method ensures that all schemas defined within BizTalk application can be expressed using the same language, XSD.

Normally the content of the message defined by a schema is not visible to the orchestrations or messaging agent. However it is possible to mark some

elements in the schema either *properties* or *distinguished fields*. In such case, the content of those elements can be accessed by the system and for example routing decisions can be based on them [17].

5.5.2 Maps

Schemas are used to describe incoming and outgoing messages, information the system would like to receive and information it is supposed to pass on. But the mere messages alone are not sufficient, unless we are building a pure routing application which only receives messages and passes them on. The goal on the enterprise integration is to get two information systems to understand one another. In order to do transformations from one type of messages to another we need a tool to define relationships between data in messages of type A and B.

In BizTalk this is done using a graphical tool, BizTalk Mapper to create maps. A map is a graphical representation of relationships between two schemas. Those relationships can be simply just direct links between elements in the schemas or more complex transformations. Latter are modeled using *functoids*, which is a basically a piece of executable code taking an arbitrary number of inputs and performing some transformation on them [6].

Maps are implemented using the standard XSLT (eXtensible Stylesheet Language: Transformations) language, but they are stored during development using BizTalk Mapper's own binary format, which stores also the graphical layout of the map. That information is transformed into an XSL document during compilation of the map. In the XSL document functoids are usually implemented as in-line code on some .NET compliant language, XSL's (or XPath's) own functions are rarely used [18].

It is also possible to create maps directly using handwritten XSL document as a transformer. This helps greatly porting applications to the BizTalk platform, since old XSL documents can be used as is. Even when a map is referencing to external XSL document, the transformation logic is compiled into the assembly. Thus it is not possible to alter the XSL document without recompiling and re-deployment [18].

BizTalk comes with a number of build-in functoids for the most usual transformation tasks. New functionalities can be implemented by directly writing in-line code on some .NET compliant language or XSLT, there is a specific functoid for embedding in-line this code into the map. The other option is to code new classes on top of the .NET's BaseFunctoid class. This approach allows creating re-usable custom functoids. Custom-made functoids can ei-

ther expose methods to be called from the map or they can offer source-code to be embedded. The latter option offers some interesting possibilities [18].

A common limitation for both of these approaches is that the code only sees elements of the source schema as strings, not as a XML-elements or tree-fragments. This limits for example accessing the parent of the element using XPath [18].

Maps may be used either within orchestration shapes to construct messages or attached to send or receive ports to perform some transformations even before the message enters the message box or after it has already left the system.

5.5.3 Orchestrations

Since maps can be attached both incoming and outgoing ports, it would be possible to create simple integration applications by using only schemas and maps. But if there is a need for a bit more complex business logic, a developer needs orchestrations, which in BizTalk are used to implement the actual functionalities needed.

Orchestrations are executable segments, which are developed using graphical editor, BizTalk Orchestration Designer instead of some formal and textual programming language [6].

Orchestration Designer uses a group of shapes to represent tasks performed by the orchestration, such as constructing messages, simple decisions, looping etc. Shapes are simply dragged from the toolbox to designer surface an connected to each other. The process is very much like flowcharting. Shapes available for the developer to use are listed in table 5.2. Unlike maps which allow developing new functoids, defining new shapes for orchestrations is not possible. On the other hand, all needed functionalities can be achieved via Expression shape which can call arbitrary code.

It is also possible to embed .NET code directly into the shapes. The language used is called XLANG/s and its syntax is much like that of C#, but their functionalities differ greatly. During compilation, the orchestration is transformed into MSIL and they are deployed like any other .NET assembly. Thus it is not possible to modify orchestrations "on the fly". Instead, if some functionalities are wished to alter, the orchestration must be un-deployed, modified, compiled and deployed again. If a more flexible approach to operations is desired, the tool called Business Rules Engine, which is not dealt with in this thesis, should be used [6].

Orchestration may be started either by receiving a message through message box database or after the orchestration is called from another orchestration. After initiation, the orchestration may run arbitrary long time, sending for example multiple requests and waiting for responses [19].

Since long-running orchestrations may spend a great deal of time waiting for responses to their requests, BizTalk Server can perform procedures called *dehydration* and *rehydration* in order to conserve resources. Dehydrating an orchestration instance means saving its into the database, freeing the memory resources it uses. BizTalk Server can do this for orchestration instances that have been idle for a while, for example waiting for a response. After the response arrives, Server will rehydrate the orchestration allowing it to continue its operations from the point where it was dehydrated. These procedures enhance servers performance and allow it to run multiple long-running orchestrations on same time with a minimal resource consumption [17].

Messages are send are received to and from the orchestration via logical ports. They should not be confused with physical ports defined earlier. While physical ports are interaction points between BizTalk Server and the world outside, logical ports are used by the orchestrations within the server to communicate with the server [17].

Logical ports are bound to physical ports during the enlistment of the orchestration thus creating subscriptions for the messages. The other option is to specify physical ports during designing the orchestration. In that case, deploying the orchestration also creates these ports [19].

5.5.3.1 Correlation

The concept of correlation and correlation sets is important when working with orchestrations performing complex communication with external entities. In such cases there may be several orchestrations of the same type running at the same time, and they can all be waiting for a same type of response for some request they have sent. How does the arriving message get to the right orchestration, since they have all a subscription on that message?

BizTalk solves this problem using correlation sets. A correlation set is a list of properties of a message that is used to select the right orchestration from the all running orchestration to receive the message. These properties can be based on both the content of the message or it's context [17].

It is possible for example to use an id of the supplier as a correlation property

when sending orders. In such case, order confirmations from the supplier get directed to the right orchestration.

5.5.4 Monitoring and interaction tools

As BizTalk Server uses its message box database to store key parts of its state, it would be possible to monitor the Server by just using some SQL queries. To make things easier for the administrator, the BizTalk contains a tool called Health and Activity Tracking (HAT), which provides a graphical interface monitoring the state of the system [19].

HAT provides tools for both performance monitoring and debugging errors. It is possible to list for example ongoing or suspended orchestrations, examine messages currently in the system and track their path or calculate statistics about the message throughput [19].

It is sometimes desirable to integrate human workforce into a business process handled by the BizTalk. With BizTalk Server 2004 this can be done with Human Workflow Services (HWS), which give interface for the orchestrations to call for the human intervention (much like invoking other orchestrations). On the client side, HWS can be implemented for example using programs in the Microsoft Office family, with which most information workers are probably familiar [6].

5.6 Summary

Microsoft BizTalk Server 2004 is an integration tool to be used both within and between organizations. It provides a framework for tasks such as receiving, processing and sending messages as well as tools for monitoring the health of the system, giving a developer a possibility to concentrate on defining the business processes.

BizTalk offers a lot more functionalities that have been covered by this thesis. Although more functionalities means more versatility, it also makes the BizTalk Server a hard to understand and comprehend when only a small set of these functionalities are needed.

Name	Purpose
Call Orchestration	Synchronously calls another orchestration.
Call Rules	Calls and executes a business rules policy
Compensate	Undoes operations that have already been performed
Construct Message	Constructs new messages. The shape contains any number of Transform shapes and/or Message Assignment shapes that actually construct the message. This shape serves only as a frame to them.
Decide	Performs simple if-then-else decisions based on a boolean condition. Causes the orchestration to branch.
Delay	Suspends the execution of the orchestration for given amount of time.
Expression	Embeds .NET code into orchestration. Can be used to method calls or variable assignments.
Group	Groups shapes together. Offers no functionalities, the shape exists only for visual purposes.
Listen	Like Decide shape but conditions for entering branches are not boolean conditions but either Delay or Receive shapes.
Loop	Used to construct while-loops to perform some tasks while some boolean condition is true.
Message Assignment	Assigns values to messages properties or distinguished fields. Can only reside inside a Construct Message shape.
Parallel Actions	Performs some tasks simultaneously, not sequentially.
Receive	Orchestration receives new messages through this shape. It is connected to a receive port.
Role Link	Creates a collection of ports to communicate with some business partner using a predefined role.
Scope	Constructs a try-catch like structure. Shapes within a scope are treated like an atomic transaction. This shape must also contain a Compensate shape to undo the effects of those shapes in case of an exception.
Send	Messages are send through send ports using this shape.
Start Orchestration	Calls another orchestration asynchronously.
Suspend	Suspends the execution of the orchestration until it is recovered using HAT. Can be used to allow administrator to react errors.
Terminate	Completely terminates the execution of the orchestration.
Throw Exception	Throws and exception in case of an error, works much like throw clause in other programming languages.
Transform	Uses a map to transform a source message to a new one. Can only be located inside a Construct Message shape.

Table 5.2: Shapes available for orchestrations in Microsoft BizTalk Server 2004 [17]

Chapter 6

Implementation

Although the insiders register and ownership register essentially show the same kind of data, there are some differences and thus there is a need for two different applications with some shared components. Because insider register was considered to be more business critical (after all, it well become mandatory) the implementation part of this thesis will cover only implementing and insider register application on top of Microsoft BizTalk Server 2004. Later it shall be referred either as the insider register application or simply as application.

In this chapter the implementation process and the structure and functionalities of the application are described. At first we present the original plan after which problems faced during the implementation process are discussed and finally the actual outcome of this project is presented.

In this chapter we refer insiders and their linkages as *owners* even if all of them do not own any shares on the company. Thus we can also speak about owner-ids as tools to identify insiders and their linkages.

6.1 Overview

The purpose of this application is to consume daily insider registry reports received from APK, to process and store them and to display user insider's and his/her linkages' holdings on some point in time, changes in those holdings as compared to some other point in time and the transactions that have taken place between those points.

It is easy to see that the three-tier architecture described in the figure 6.1 can

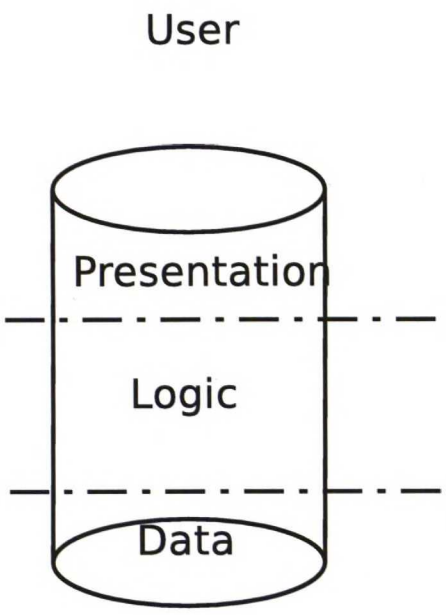

Figure 6.1: Three tier architecture [7]

be applied to this problem. The data-tier handles processing the incoming data and storing it, logic-tier computes stored data for the presentation-tier to display. It's worth noting that presentation tier is not to be affecting the stored data, its only purpose is to display it and all modifications to the data are done by the data-tier itself based on the data it receives.

During implementation phase the implementation of the presentation-tier was excluded from this thesis and transferred to a different developer. Also the logic-tier proved to be quite simple, since the business logic in this application is mainly calculation of changes in amounts and could be implemented easily and fast as a simple .NET webservice using XSL technologies. Since that was done outside the BizTalk Server, it is beyond the scope of this thesis. Thus this chapter concentrates on the data-tier.

6.2 Original plan

The data-tier was supposed to be implemented as a single BizTalk orchestration which would decompose an insider report into three different XML documents to be stored on the disc and served to the presentation tier by logic tier. These documents would be day's snapshot of ownerships, day's transaction-list and the list of insiders and their linkages.

Before entering orchestration the original document would go through pipeline and inbound map that would transform it into internal format which is then processed by the orchestration. Resulting documents would be stored to the disc using a send port. Adding new data-sources would only mean creating a new receive port to receive the data and creating a map that transforms that data to the internal format.

The Finnish insider data from APK was meant to be the first business case. In the next section problems faced during the implementation are presented along with the actions performed in order to solve them.

6.2.1 Problems

Probably no software implementation project goes by without any problems faced along the way. In this section we shall present some of the problems faced during the implementation.

6.2.1.1 Identifying an owner

One problem with the original insider register was that the data contained no unique identifiers to be used to identify one owner from report to report. The original implementation operated on *namestrings* and mapped them to artificial *owner-ids*. A namestring is a representation of the insider's or owner's real name. The problem is that it was not unambiguous, for example mister John Richard Doe can be represented by namestrings: "Doe John", "Doe John Richard", "John Doe", "John Richard Doe", "Doe John R" and so on. This is especially true with juridical persons (companies, foundations, investment funds) that have a different name on different languages since the APK's system seems to be picking the language and namestring used by random.

Thus the original implementation used to map namestrings to owner-ids, allowing "Doe John" and "Doe John Richard" to be identified as a same person. This mapping had to be done by a human, since it required a large amount reasoning.

The new implementation was supposed to be using some other identifier instead of the namestrings. That would have been social security number for natural persons and business identity code for juridical persons which both identify the person unambiguously and would be available in the APK's data.

The Finnish law takes social security numbers quite seriously. Thus using

data containing them would cause some extra demands considering the safety and confidentability of the data (business identity codes are easier in that sense, since they are public information anyhow): social security numbers should not be displayed in public or even "unnecessarily stored" [26]. The easiest solution would be to use for example some hashing algorithm before the data even enters the system to hide the social security numbers to still maintain the unambiguous identity.

The greatest problem with social security numbers was however financial, not a technical one. APK priced the data so that a report with ids was over ten times more expensive than the report with only namestrings [27]. After consideration is was decided to continue using namestrings as an identifier, since the data was needed daily and it would be unlikely that the clients would be willing to bear the extra costs of using the more expensive data.

The owner identification process needs a separate orchestration which matches the namestrings in the data with owner-ids. This will also cause a need for human interaction in case of an unidentified namestring. That namestring may either be a new namestring referring to an already known owner or a namestring identifying a completely new owner. A back-office person would be needed to handle these issues.

6.2.1.2 Fetching the data

The new insider register was supposed to fetch its data from the source by itself. This would be achieved using the APK's new extranet and fetching reports via HTTP-protocol. Of course some other data providers might still need manual interaction.

When the APK's extranet was opened for pilot testing it was found out that the automated fetching of the data would not be possible. The extranet was a complex set of forms to navigate through and the reports would have to be ordered before they could be downloaded. The ready reports could not be referred by any direct URL, even they had to be found via forms. The extranet was a *user interface* not a *technical interface* on the core.

The situation was then re-evaluated. Using the the out-of-the-box HTTP adapter of BizTalk Server was not possible. Developing a custom HTTP adapter that would be able to download data from the APK's extranet would perhaps be possible but it would take considerable amount of time. Because the extranet is a user interface, APK could alter it at any time, without notice, which would cause the adapter to become useless. When asked about a possibility to obtain data through some technical interface in the future,

APK's answer was that "it will be taken into consideration". Thus after consideration, it was decided to use BizTalk's FILE adapter instead, to consume reports from filesystem and the development automated data fetching was postponed until APK offers a webservice or some other more technical interface. Because the namestrings already require a back-office employee, that same person may as well handle downloading the reports and feeding them into the process via a "drop-box", a specific folder in the filesystem. So data was inserted into the system via BizTalk's FILE adapter instead of HTTP adapter, as was the original plan.

6.2.1.3 The quality of the data

The first problem encountered with the APK's data was their inability (or unwillingness) to provide an XML schema that describes the format. Instead only a verbal description and some sample reports were provided.

The BizTalk Server requires an XSD schema for each type of documents it is supposed to process and writing an XSD by hand is quite time consuming, especially when the structure of the document is quite complicated like in this case. Luckily the Microsoft Visual Studio contains a wizard that is able to generate a schema from a sample document. The wizard is of course not perfect, but the schema it created sufficient after some modifications and corrections.

After the schemas was created, it revealed some other limitations of the data. First one being that APK uses a format dd.mm.yyyy to represent dates, even though the World Wide Web Consortium recommends using a format yyyy-mm-dd to represent dates [38].

The third problem was related to the natural persons as linkages. Since the law does not require their names to be published, APK's data does not contain any names for them. They are just represented as entries in the linkage-list without any identifier. Because the identifier orchestration needs a namestring, some arbitrary namestring was needed to be created for those linkages.

Both of the problems above were solved by a transformation from the original APK schema to an internal schema. During that transformation the dates were converted to a correct format and unnamed linkages were given an arbitrary namestring, consisting of the name of the insider, his/her relationship to the linkage and an order number, for example: "John Doe Child 1". In addition to that, the transformation dropped some unneeded information from the document.

6.2.1.4 Historical data import

The basic procedure only handles messages containing a snapshot of the situation on one day. When setting up new client, also historical data is required, since the law demands for showing past transactions from a period of one year. It would be possible, but both time consuming and expensive to fetch a daily report for each day of a year when new client is being set up.

Since APK also offers historical reports, it was decided to create an method for consuming one of such reports and producing a daily report for each day it contains. Thus all the historical data can be imported from a single document.

6.3 Actual implementation

The previous section described the original implementation plan and the problems encountered while trying to follow it. In this section we present the structure and functionalities of the actual application implemented.

The application was published and launched at the end of February 2006.

6.3.1 Overview

The figure 6.2 shows the path of a insider registry report through the system and the parts that make up the entire insider registry application. The first two stages, **ProcessInsiders** and **OwnerIdentifier** were implemented as orchestrations using BizTalk Server 2004 and are the main concern of this thesis. **DataStorage**, illustrated as a directory structure, represents the location in the file system where the data files are stored after processing. **WebFrontEnd** and **WebBackEnd** are the web interface offered to the user and the web service serving data to that interface. **OwnerDataBase** refers to the database containing ownersids and corresponding namestrings. The entire system illustrated in the figure 6.2, notwithstanding the WebFrontEnd, resides in a single computer.

The figure does not cover importing history, which is mainly done by a single orchestration feeding its results to the system.

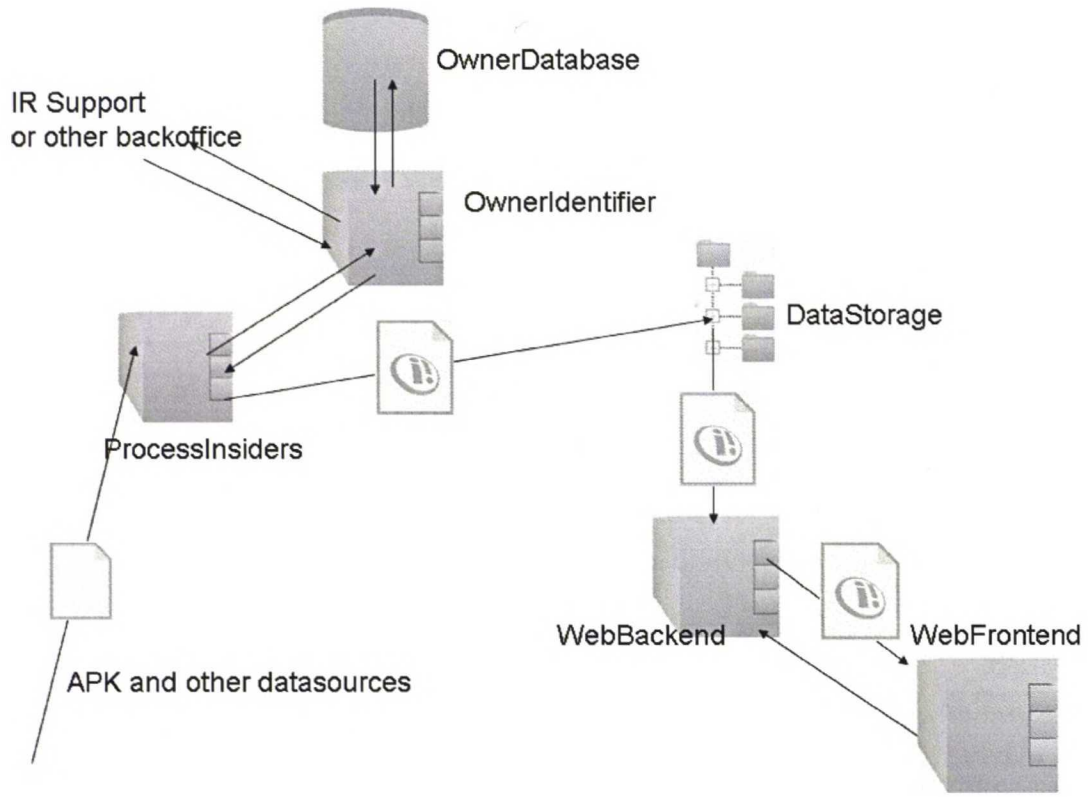

Figure 6.2: The final architecture of the insider register application

Name	Purpose
OwnerIdentifier	Matches namestrings with owner-ids. Sends the unidentified ones to the back-office worker for manual processing.
ProcessInsiders	The main orchestration, transforms the incoming data into separate documents and sends them to the storage.
ProcessHistory	Takes as an input a report for longer period of time and constructs a corresponding amount of daily reports and sends them to ProcessInsiders orchestration.

Table 6.1: Orchestrations used in the insider register application

6.3.2 The flow of the document

When a new message enters the application via the FILE adapter, it is first picked up by the ProcessInsiders orchestration. This orchestration performs some initial sanitization for the data, after which it is send to the OwnerIdentifier orchestration for identification of the owners and other entities.

After the message returns from OwnerIdentifier (now containing ids for all the owners) it is decomposed into three separate messages: the first contains information about insiders and their relationships, the second deals with ownerships of securities on that day and the third contains all the transactions performed that day.

Those three messages are then stored (send) using FILE adapter to the right folder in the filesystem for later use.

Historical data enters the system through FILE adapter, but via a different folder. It is processed by ProcessHistory orchestration which constructs corresponding amount of daily reports which are then given to the ProcessInsiders orchestration via the same folder it receives its "normal" inputs. The receive-location of one port uses the same folder as the other send port.

6.3.3 Orchestrations in detail

When using BizTalk most of the functionalities are implemented using orchestrations. In this section we go through the three orchestrations, that are listed in table 6.1, of the insider register application in detail.

6.3.3.1 ProcessInsiders

This is the main orchestration, responsible for consuming incoming messages and storing the outcome to the disk. The processing is done in three phases: initial conversion, identification and storage.

First the incoming message is transformed into internal format. This is done by a map which transforms dates into correct format, gives names to unnamed linkages and so on, as described in the section 6.2.1.3. Most of these tasks are performed by map's custom made functoids, which are actually .NET objects that inherit the BaseFunctoid class.

As the result we get a message in which each owner has some kind of namestring identifying it. This message is then used as a parameter when calling the OwnerIdentifier orchestration. Functionality of that orchestration is explained in section 6.3.3.2.

The message returned by the OwnerIdentifier orchestration has the same structure (and thus same XML schema) as the message it was given. The only difference it that now each insider and linkage has an id attached to the namestring.

In the third phase the message is used as a basis for constructing three messages, containing insiders, ownerships and transactions. This is done by maps. BizTalk Mapper proved to lack capabilities to map hierarchal data to same level: both insiders and their linkages may own securities and we wanted a same kind of element to represent both of these situations. Since BizTalk's Looping functoids didn't manage to handle this, most of the logic in these maps was done using Scripting functoid and in-line XSLT. It was noticed that even though orchestration's Expression shape provided developer with IntelliSense, a helper tool for code generating [16], Scripting functoid did not. This made writing custom code into map a bit more difficult since normal helper functionalities were not available.

After the three messages have been created, they are stored to disk using BizTalk's *dynamical ports*, that is ports, that can be given the destination folder during run time. They were needed since we wanted to organize the data into folders by client's name.

6.3.3.2 OwnerIdentifier

This orchestration is called by the previous one in order to give owners their ids. The orchestration is called by the ProcessInsiders orchestration and

given an SIREReportInternal as an argument. The orchestration consists of a single loop-shape which invokes IdentifyOwners map over and over until resulting SIREReportInternal message has all of its owners identified.

In case there are unidentified owners, an UnidentifiedOwners message containing the unidentified namestrings is constructed and send through a port to back-office for manual processing. After that the orchestration needs wait for the back-office to either map the namestrings to existing owners or create new owners before trying to identify owners again. The first attempt to provide this behavior was using the Suspend shape to put the orchestration on hold during the manual operations.

The problem with the first approach was that the back-office worker needed to BizTalk's Health and Activity Tracking (HAT) tool in order to resume the suspended orchestration. This added and unnecessary step to the manual process and using HAT is not something that an information worker working with BizTalk should need [15]. Even further, the Suspend shape is intended to stop orchestration for debugging in extraordinary situations [15], which was not the case here. Thus using suspension as a solution was rejected.

Suspension was replaced with a Receive shape, which shared a Correlation Set with the Send shape sending the unidentified names to back-office. The correlation set related to the company's name that was as an attribute in the error message. Hence sending for example the generated error message back would trigger the orchestration to continue looping.

After exiting the loop the resulting SIREReportInternal is send back to the calling orchestration.

The map identifies owners using database lookup functoids, which search in the owner database for an owner-id that corresponds the namestring in question. Those functoids have limited searching capabilities, for example they can only retrieve one data row from the database, no matter how many rows the dataset resulting from the query returns [15] This was not a problem for the NameResolver, but the localization manager that was originally intended to be implemented as BizTalk orchestration proved to be impossible to implement without developing custom functoids for fetching multiple rows of data. Thus the localization manager was implemented as a separate service. Hence also the localization information was stored into an XML file, instead of the database, as was the original plan. Hence handling the localizations was moved outside the BizTalk server and thus it will not be dealt with in this thesis.

6.3.3.3 ProcessHistory

This orchestration was added afterwards in order to import historical data more efficiently. It contains two maps and a loop, which goes through an historical report day by day, constructs a daily report and sends it forward to the ProcessInsiders orchestration.

All maps used by this orchestration were completely written in XSLT abolishing the BizTalk Mapper completely, which proved to be quite effective way. To achieve the desired functionalities. This kind of development is actually quite usual.

Chapter 7

Evaluation

In this chapter we shall evaluate the results of the implementation process described in chapter 6. The evaluation is divided into three parts. In section 7.1 the applicability of the concept of e-business into this implementation is questioned. After that, in section 7.2, the tool used, the Microsoft BizTalk Server, is evaluated as a development platform. Section 7.3 evaluates the results in respect to the requirements and criteria defined in 2.2.2. Finally, some conclusions are made.

7.1 Insider register as an e-business solution

Business integration is a difficult process. It becomes even more difficult when the other party has no significant desire to be integrated. In this case APK has a double role both as data provider and a competitor. Quite naturally, their focus is on selling refined data, their own application, instead of providing competitors with raw data.

As has been said, APK was not able to provide XML schema (or any other formal definition) of their data. Generally speaking, it is good for the integration partner to know the language, other partner is speaking. In this case, the specs of the language were generated by a wizard by using a sample of data and a written data specification.

The section 6.2.1.3 was about the quality of the data. It seems that the data provider not only lacks the ability to provide XML schema, it also lacks the ability to provide decent XML. In addition to using strangely formatted dates, APK's data specifications contain anomalies like representing booleans with an element which has content 'X' when the value is true otherwise

being empty [29]. These are of course only technical details, but nevertheless they are a cause for much extra work, since the data must be "cleaned" and "rationalized" before it can be used.

7.2 BizTalk Server 2004 as a development platform

During this project, many limitations of the BizTalk Server were noticed. Most of these are related to BizTalk Mapper, which has also been widely criticized elsewhere.

BizTalk Server does provide "out of the box" many features and functionalities commonly needed for information systems integration. This is quite time-saving, since there is no need to "reinvent the wheel". On the other hand, BizTalk has limitations, many of which have been encountered during this project.

Immutability of messages, as described in section 5.4 was a source of much confusion. Due immutability, values of properties and distinguished fields of a message could not be altered after the creation of the message. Due this, it was for example not possible to use a distinguished field of a message as a loop variable. Instead, that message had to be reconstructed by a simple identity transformation for every round. After a new message was constructed, its distinguished fields could be altered.

Due the limited scope and needs of this project, many aspects of BizTalk Server, like transactions, Human Workflow Services, exceptions and compensations were not throughly examined. Hence, the perspective gained on BizTalk was left somewhat shallow and the limitations of the environment were the most dominant experience.

Perhaps BizTalk is not so well suited for developing actual information processing systems, like this one. Its potential could be far better utilized as routing and transformation engine. In such scenario BizTalk's role would be simply to route messages between information systems, which contain all the business logic needed.

7.3 Evaluation against defined criteria

The criteria to be used to evaluate the solution were discussed in section 2.2.2. Those were minimal need for human interaction and easy addition of additional data sources. Next we will evaluate the solution with respect of those criteria. We shall see, that both of these criteria were failed to meet.

7.3.1 Minimal need for human interaction

As can be seen from chapter 6, the need for human interaction could not be totally abolished. Specifically there were two tasks that needed some external intervention: fetching data and resolving identification issues.

Both of these issues are related to the nature of the data and its provider. Like it was noted in the section 7.1, the data provider can or will not provide a decent technical interface for fetching data, hence fetching must be done by hand. If the data was available through for example a web service interface, fetching could be automated. If that kind of interface becomes available some day, plugging it into the application is relatively simple: only the adapter used in the receive location has to be changed.

Also the need for owner identification can be seen to be caused by the provider. Since data containing reliable identifiers is not available on reasonable price, the namestrings have to be used. And because the provider fails to ensure stability of those namestrings (as was discussed in the section 6.2.1.1) a human information worker is needed in the identification process. Even more, it require a Finnish-speaking information worker, who is able for example to recognize Finnish abbreviations used in the names of the companies and so on.

Thus the first criteria fails to be met, but the nature of human interaction needed has been radically changed. The new application does not need a software developer to run daily operations. Instead, it requires an information worker or a back-office employer. This can be seen as an improvement.

7.3.1.1 Easy addition of the new data sources

In addition to APK, no other data providers were considered during this project.

As was seen when implementing the history import orchestration, data in new format is quite easy to tap into, if it can be transformed to the same

format as the raw format from APK, which, unluckily, is somewhat unusable format. From this point of view, it would have been better to implement ProcessInsiders orchestration to consume data in some internal format and then develop a set of orchestrations that transform incoming data into that format. The current implementation consumes APK's format and does the transformation itself.

Thus the second criteria can be considered as partly achieved. New data-sources can be added, but addition is not so easy as desired.

7.3.1.2 Conclusions

The new insider register application can process and provide the data required by the new legislation. This is of course the most important achievement. The additional criteria that were set in the beginning of project were on the other hand not completely achieved: human interaction is still needed but the nature and amount of it have dramatically changed and the new data sources can be added, but the application is not designed specifically to support that.

Chapter 8

Conclusions

As a result of this thesis, a new investor relations service application was developed and launched. Although the road taken was paved with drawbacks and problems, related to both tools used and the business partners operated with, the development can be considered successful. The application performs the desired tasks, even though both of the criteria set in the section 2.2.2 were failed to meet, as explained in the section 7.3.

The viewpoint taken in this thesis was that of the e-Business solution, or business-to-business integration. Some problems encountered were caused by the other integration party, the Central Securities Deposit, and its lack of co-operation, as stated in the section 7.1. XML has been the great enabler of e-Business, but as it was seen, it can be used incorrectly, either due to the purpose or due to the lack of expertise. The problems encountered can be summarized as follows: the business to business integration project is reasonable only if both parties really want to be integrated. This conclusion seems a bit trivial one, but worth mentioning.

On the other hand, during the implementation a tool designed mainly as an integration platform was used to perform complicated tasks related to business logic. Even though BizTalk managed to do everything needed, most of the tasks would have been easier to implement using some other platform. Implementing integration projects with BizTalk is easy, if only the components included are used, the interfaces offered for user-defined components were proved to be bit too limited. BizTalk is clearly an integration platform designed for business environments where processing documents is a support process supporting the main business. Thus there is no need for complicated processing. In the investor relations application like this one, the documents and their processing are the business.

On the other hand, the BizTalk Server contains numerous components and properties, all of which were not covered by this thesis. It is possible that some of these features would have easily tackled the problems encountered, if they only were known.

One goal set for this thesis was evaluating BizTalk as a future platform for investor relations services. The results of the evaluation are not that encouraging. Given the current level of BizTalk expertise within the organization, the platform cannot be recommended for further development projects.

Bibliography

[1] ABELSON, H., SUSSMAN, G. J., AND SUSSMAN, J. *Structure and Interpretation of Computer Programs*, 2nd ed. The MIT Press, 55 Hayward Street, Cambridge, MA 02142, 1996.

[2] BERNERS-LEE, T., FIELDING, R., AND MASINTER, L. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Draft Standard), Jan. 2005.

[3] BRADLEY, N. *The XML companion*, 3rd ed. Pearson Education Limited, 2002.

[4] BREALEY, R. A., AND MYERS, S. C. *Principles of Corporate Finance*, 7th ed. McGraw-Hill Inc, 2003.

[5] BURT, D. N., DOBLER, D., AND STARLING, S. L. *World Class Supply Management - The Key to Supply Chain Management*, 7th ed. McGraw-Hill Inc, 2003.

[6] CHAPELL, D. *Understanding BizTalk Server 2004*. Microsoft Corporation, Feb 2004. `http://go.microsoft.com/fwlink/?LinkId=21313` accessed Oct 26th 2005.

[7] CLEMENTS, P., AND ROGERS, F. Three tier software architectures - software technology roadmap. Tech. rep., Carnegie Mellon Software Engineering Institute, 2000. `http://www.sei.cmu.edu/str/descriptions/threetier.html` accessed Dec 20th 2005.

[8] COPELAND, K. W., AND HWANG, C. J. Electronic Data Interchange: Concepts and Effects. In *Proceedings of the Commerce track of the INET'97 Conference* (June 1997), Internet Society. `http://www.isoc.org/inet97/proceedings/C5/C5_1.HTM` accessed Apr 18th 2006.

[9] DUERST, M., AND SUIGNARD, M. Internationalized resource identifiers (IRIs). RFC 3987 (Draft Standard), Jan. 2005.

[10] FINNISH FOUNDATION FOR SHARE PROMOTION. Stock exchange vocabulary. `http://www.porssisaatio.fi/default.aspx?path=4;164;211`. accessed Sep 10th 2005.

[11] GOLDFARB, C. F., AND PRESCOD, P. *Charles F. Goldfarb's XML Handbook*, 5th ed. Prentice Hall, 2004.

[12] KAY, M. *XSLT Programmer's Reference*. Wrox Press Ltd, Arden House, 1102 Warwick Road, Acock's Green, Birmingham B27 6BH,UK, Jul 2000.

[13] LAESVUORI, H. RosettaNet implementation using Microsoft BizTalk with emphasis on exception handling. Master's thesis, Helsinki University of Technology, 2003.

[14] LEE, D., AND CHU, W. W. Comparative analysis of six XML schema languages. *SIGMOD Rec. 29*, 3 (2000), 76–87.

[15] MICROSOFT CORPORATION. *BizTalk Server 2004*. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanch%or/html/anch_Biztalk2004.asp` accessed Feb 19th 2006.

[16] MICROSOFT CORPORATION. *.NET Developer's Guide*. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanch%or/html/netdevanchor.asp` accessed Jan 22nd 2006.

[17] MICROSOFT CORPORATION. *Developing E-Business solutions using Microsoft BizTalk Server 2004*, 1st ed., 2004. Microsoft Course Material.

[18] MICROSOFT CORPORATION. *Microsoft BizTalk Server 2004 Programming Guide*, 2005. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sdk/ht%m/ebiz_prog_intro_onhk.asp` accessed Feb 19th 2006.

[19] MILNER, M. *BizTalk Server 2003: A messaging engine overview*. Microsoft Corporation, May 2005. accessed Oct 24th 2005.

[20] NAGPAL, A., AND PITLAK, J. *ALE,EDI and IDoc technologie for SAP*, 2nd ed. Prima Tech, 2001.

[21] NAKHIMOVSKY, A., AND MYERS, T. *Professional Java XML programming with Servlets and JSP*. Wrox Press Ltd, Arden House, 1102 Warwick Road, Acock's Green, Birmingham B27 6BH,UK, 1999.

[22] NAVARRO, A., WHITE, C., AND BURMAN, L. *Mastering XML*. Sybex, 2000.

[23] PELKONEN, H. *Yritysten välinen tiedonsiirto - EDI*. Sähkö- ja elektroniikkateollisuusliitto, 1997.

[24] SELIN, M. Metso ja Alma pokkasivat verkkokultaa. *Talouselämä* (Jan 2006). `http://www.talouselama.fi/doc.te?f_id=837192` accessed Jan 21st 2006.

[25] STÅHLBERG, K. Sisääpiirisääntely liikkeeseenlaskijan sisäpiiriläisen kannalta. *Defensor Legis*, 5 (2000), 731–747.

[26] THE DATA PROTECTION BOARD. Henkilötunnuksen käsittely henkilötietolain mukaan. `http://www.tietosuoja.fi/1974.htm`, Nov 2004. accessed Feb 19th 2006.

[27] THE FINNISH CENTRAL SECURITIES DEPOSITORY. List of prices of the insider data reports. Unpublished, 2005.

[28] THE FINNISH CENTRAL SECURITIES DEPOSITORY. Netsire. `http://www2.apk.fi/NewNetSire/` accessed Jan 4th 2006, 2005.

[29] THE FINNISH CENTRAL SECURITIES DEPOSITORY. XML report specifications. Unpublished, 2005.

[30] THE FINNISH FINANCIAL SUPERVISION AUTHORITY. Standard 5.3 Declaration of insider holdings and insider registers, Sep 2005.

[31] THE STATE OF FINLAND. The securities market act. `http://www.finlex.fi/fi/laki/ajantasa/1989/19890495`. accessed Apr 14th 2006.

[32] WORLD WIDE WEB CONSORTIUM. *Namespaces in XML*, Jan 1999. `http://www.w3.org/TR/REC-xml-names/` accessed Feb 19th 2006.

[33] WORLD WIDE WEB CONSORTIUM. *XML Path Language (XPath)*, 1st ed., Nov 1999. `http://www.w3.org/TR/xpath` accessed Feb 19th 2006.

[34] WORLD WIDE WEB CONSORTIUM. *XSL Transformations (XSLT)*, 1st ed., Nov 1999. `http://www.w3.org/TR/xslt`, accessed Feb 19th 2006.

[35] World Wide Web Consortium. *Extensible Markup Language (XML) 1.1*, Apr 2004. `http://www.w3.org/TR/2004/REC-xml11-20040204/` accessed Feb 10th 2006.

[36] World Wide Web Consortium. *Namespaces in XML 1.1*, Feb 2004. `http://www.w3.org/TR/xml-names11/` accessed Feb 1st 2006.

[37] World Wide Web Consortium. *XML Schema Part 0: Primer*, 2nd ed., Oct 2004. `http://www.w3.org/TR/xmlschema-0/` accessed Feb 1st 2006.

[38] World Wide Web Consortium. *XML Schema Part 2: Datatypes*, 2nd ed., Oct 2004. `http://www.w3.org/TR/xmlschema-2/` accessed Feb 1st 2006.